Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Previously presented) A method for managing flow of messages between two software modules, said method comprising:

(a) determining whether a first message can be propagated to or from a first synchronization queue associated with one of the two software modules, by a first thread running on a first processor, while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

(b) propagating the first message to or from the first synchronization queue while allowing the second thread to propagate the second message between the two software modules when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules.

2. (Previously presented) A method as recited in claim 1, wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first synchronization queue.

3. (Previously presented) A method as recited in claim 1, wherein the method further comprises:

(c) blocking the second thread from propagating the second message between the two software modules when said determining (a) determines that the first message cannot be propagated by the first thread.

4. (Previously presented) A method as recited in claim 2, wherein said blocking (c) of the second thread from propagating the message between the two software modules is achieved by providing a lock which can be acquired by the first thread.

5. (Previously presented) A method as recited in claim 1, wherein the method further comprises:

setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules; and

setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

6. (Previously presented) A method as recited in claim 1, wherein said determining (a) comprises:

(a1) determining whether an event is being processed or is pending to be processed; and

(a2) determining whether a thread-count for the first processor is zero, and

wherein said determining (a) determines that the first thread can propagate the first message without blocking the second thread from propagating the second message when said determining (a) determines that no events is being processed or pending and the thread-count for the first processor is zero.

7. (Cancelled)

8. (Cancelled)

9. (Cancelled)

10. (Previously presented) A method as recited in claim 1, wherein the two software modules are implemented in a stack as STREAMS modules.

11. (Previously presented) A computer system comprising:

a plurality of processors;

first and second software modules, the second software module having a main queue suitable for storing messages and an auxiliary queue suitable for storing messages that are not stored in the main queue; and

a propagation controller for propagating messages between the first and the second software modules, the propagation controller operating to enable at least two processors of said plurality of processors to concurrently propagate messages to or from the auxiliary queue of the second software module.

12. (Original) A computer system as recited in claim 11,

wherein the propagation controller comprises:

a thread-count for one of said plurality of processors; and

a queue count for the auxiliary queue.

13. (Original) A computer system as recited in claim 12,

wherein the auxiliary queue is a synchronization queue and the queue count is a synchronization queue count.

14. (Original) A computer system as recited in claim 11, wherein the at least two processors of said plurality of processors concurrently propagate a message from the first software module to the auxiliary queue of the second software module.

15. (Original) A computer system as recited in claim 11, wherein the at least two processors of said plurality of processors concurrently propagate a message from the first software module to the main queue of the second software module.

16. (Previously presented) A computer system as recited in claim 11, wherein the first and second software modules are implemented in a stack as STREAMS modules.

17. (Previously presented) A computer readable media including computer program code for managing flow of messages between two software modules, said computer readable media comprising:

computer program code for determining whether a first message can be propagated by a first thread running on a first processor to or from a first synchronization queue of one of the two software modules while allowing a second thread running on a second processor to propagate a second message between the two software modules; and

computer program code for propagating the first message to or from the first synchronization queue while allowing the second thread to propagate the second message between the two software modules when said computer program code for determining determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules.

18. (Previously presented) A computer readable media as recited in claim 17, wherein the first and second threads concurrently propagate respective portions of the first and second messages to or from the first synchronization queue.

19. (Original) A computer readable media as recited in claim 18, wherein the computer readable media further comprises:

computer program code for setting a first indicator for the first processor to indicate that the first processor is propagating when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two layer software modules.

computer program code for setting the first indicator for the first processor to indicate that the first processor is not propagating when said propagating (b) has propagated the first message between the two software modules.

20. (Original) A computer readable media as recited in claim 19, wherein the two software modules are implemented in a stack as STREAMS modules.

21. (Previously presented) A method as recited in claim 1, wherein the second thread also propagates the second message to or form the first synchronization queue.

22. (Previously presented) A method as recited in claim 1, wherein the second thread propagates the second message to or form one of the first synchronization queue and a second synchronization queue, the second synchronization queue being associated with one of the software modules.

23. (New) A method for managing flow of messages between two software modules, said method comprising:

(a) determining whether a first message can be propagated to or from a first synchronization queue associated with one of the two software modules, by a first thread running on a first processor, while allowing a second thread running on a second processor to propagate a second message between the two software modules, wherein said determining determines whether said first and second messages can safely be propagated without compromising data integrity; and

(b) propagating the first message to or from the first synchronization queue while allowing the second thread to propagate the second message between the two software modules when said determining (a) determines that the first message can be propagated by the first thread while allowing the second thread to propagate the second message between the two software modules, wherein said propagating operates to safely propagate said first and second messages without compromising data integrity.